Amendment to the Specification

Please add the following paragraph before paragraph [0001] and FIELD OF INVENTION:

[000.5] Cross-reference is made to the following related patent applications, which is assigned to the same assignee as the present application:
U.S. Patent Application No. 10/750,066 filed on December 30, 2003, titled "System and Method Employing Bytecode Modification Techniques For Tracing Services Within An Application Server" by Gregor K. Frey and Nikolai G. Nikolov. U.S. Patent Application No. 10/749,617 filed on December 30, 2003, titled, "Execution of Modified Byte Code for Debugging, Testing and/or Monitoring Of Object Oriented Software" by Nikolai G. Nikolov. U.S. Patent Application No. 10/749,686 filed on December 30, 2003, titled, "Registration Method For Supporting Bytecode Modification" by Nikolai G. Nikolov and Mario Kabadiyski. U.S. Patent Application No. 10/750,050 filed on December 30, 2003, titled, "Modification and Execution of Bytecode For Debugging, Testing and/or Monitoring Virtual Machine Based Software" by Gregor K. Frey and Nikolai G. Nikolov. U.S. Patent Application No. 10/750,067 filed on December 30, 2003, titled, "Byte Code Modification for Testing, Debugging and/or Monitoring Of Virtual Machined Based Software" by Nikolai G. Nikolov and Mario Kabadiyski. U.S. Patent Application No. 10/749,757 filed on December 30, 2003, titled, "Application Tracing Service Employing Different Levels of Precision for Modifying Bytecode" by Nikolai G. Nikolov. U.S. Patent Application No. 10/750,160 filed on December 30, 2003, titled, "Interfaces and Methods Employed Within A Bytecode Modification System" by Nikolai G. NIkolov. U.S. Patent Application No. 10/749,740 filed on December 30, 2003, titled, "Modified Classfile Registration With A Dispatch Unit That

Is Responsible For Dispatching Invocations During Runtime Execution Of Modified Bytecode" by Nikolai G. Nikolov.

Please replace paragraph [0003] with the following amended paragraph:

[0003] As a classic instance of the benefit of the Java virtual machine with respect to Internet usage, a first ~~PC~~ personal computer (PC) that is powered by an Intel processor may download from the Internet the same Java bytecode instructions as a second PC that is powered by a PowerPC processor. Here, the first PC's Java virtual machine converts the Java bytecode into instructions that are specific to an Intel processor while the second PC's Java virtual machine converts the same Java bytecode into instructions that are specific to a PowerPC processor. Thus, through the use of Java bytecode and processor specific Java virtual machines, an Internet server is able to maintain only a single type of code (the Java bytecode) without concern of client compatibility.

Please replace paragraph [0118] with the following amended paragraph:

[0118] **Figure 10a** shows an exemplary system comprised of an application server 1010 and a database server 1020. Application components 1011 executed on the application server 1010 may include, by way of example, presentation logic and business logic components. Within a J2EE environment, the presentation logic may contain servlets and Java Server Pages ("JSP") interpretable by browsers on clients 1000 (JSPs/Servlets generate ~~HTML~~ hypertext markup language (HTML) which is interpreted by browsers), and the business logic may include Enterprise

Java Bean ("EJB") components which are supported by EJB containers. However, as previously mentioned, the underlying principles of the invention are not limited to a Java implementation.

Please replace paragraph [0102] with the following amended paragraph:

[0102] As mentioned above with respect to **Figure 4a,** unique plugins may be developed based on the specific needs of each user. By way of example, **Figure 8** illustrates an embodiment in which three different plugins are used: an application trace plugin 810; a user-configurable plugin 820; and a distributed statistical records ("DSR") plugin 830.

Please replace paragraph [0126] with the following amended paragraph:

[0126] **Figure 12** illustrates one embodiment of the invention in which the bytecode modifier 452 modifies a particular set of methods 1211-1232 to trace program flow within a J2EE engine 1250. For example, HTTP requests transmitted by a Web-based client 1200 (e.g., a client with a Web browser) are processed by input/output method invocations 1227-1228 of HTTP logic 1201. Communication between HTTP logic 1201 and servlet/JSP components 1202 is accomplished via method invocations 1229, 1230 and 1231, 1232, respectively. Other method invocations illustrated in Figure 12 which represent entry/exit points between different J2EE services include method invocations 1213, 1214, 1216 and 1215 between enterprise Java bean ("EJB") components 1203 and servlet/JSP components 1202; method invocations 1233, 1234 and 1219, 1220 between servlet/JSP components 1202 and Java connector ("JCo") components 1204; method invocations 1221 and 1222 between JCo components 1204 and an external

system 1207; method invocations 1211, 1212 and 1223, 1224 between servlet/JSP components 1202 and Java database connectivity ("JDBC") components 1205; and method invocations 1225, and 1226 between the JDBC components and a database 1206; and method invocations 1217 and 1218 between EJB components and a non-web client 1250 (e.g., via RMI or RFC transactions).

Please replace paragraph [0138] with the following amended paragraph:

[0138] According to the methodology of **Figure 16,** modifications are inserted that are representative of additional instructions (e.g., a block of instructions) that invoke the dispatch unit at positions in the collection of objects that represent the following method locations: 1) a method entry point 1601 (e.g., as represented by modification 1504 of **Figure 15b);** 2) a method exit point 1602 (e.g., as represented by modifications 1505, 1506 of **Figure 15b);** and, 3) the end of a method's instructions for purposes of introducing try-catch block instructions 1603 (e.g., as represented by modification 1507 of **Figure 15c).**

Please replace paragraph [0153] with the following amended paragraph:

[0153] Even though the dispatch unit 1702 can deduce the proper numeric naming scheme simply from knowing the order of the methods, explicit character strings are sent to the dispatch unit 1702 to also assist the dispatch unit in the updating of its method/plug-in module correlation scheme. **Figure 18** helps explore this aspect in more detail. Figure 18 shows a registering classfile 1801 and a dispatch unit 1802.

Consistent with the preceding discussion concerning **Figure 17**, note that the registering classfile 1801 can be regardred as sending 1803 the class names and method names 1805 in character string format (e.g., "Sales", "GetMax", "UPDATE ACCOUNTING", "UPDATE BILLING"). In response, the dispatch unit's registration method 1807 refers to a plug-in pattern 1811 to identify which plug-in modules are to be applied to which methods 1808. The plug-in pattern, in an embodiment, includes a list, for each plug-in module, of every method (e.g., by class name and method name in character string format) it is expected to treat.

Please replace paragraph [0112] with the following amended paragraph:

[0112] **Figure 9c** illustrates the invocation tree 900 (initially illustrated in **Figure 9a**) within the context of the exemplary GUI. The invocation tree is shown within the second window 946 in response to user selection of an "Invocations" tab from the set of tabs 944. **Figure 9d** illustrates information related to the different classes affected by the bytecode modification of the calculator application, arranged in a logical hierarchy 947 in response to user selection of the "Classes" tab from the set of tabs 94 944. Finally, **Figure 9e** provides an object view 948 of the modified calculator application in response to user selection of an "Objects" tab from the set of tabs 944.

Please replace paragraph [0123] with the following amended paragraph:

[0123] **Figures 11 a-b** illustrate another embodiment of the invention in which distributed statistical information is collected using the bytecode modification techniques described herein. In **Figure 11a-b,** the application components 1111 within

the application server 1110 communicate with and/or exchange data with an external system ~~1120~~ 1130 (e.g., such as an R3 system designed by SAP AG) as opposed to a database server as in **Figures 10a-b.** However, similar principles apply. In this embodiment, For example, the client's 1100's initial request triggers a first method invocation 1101 within the application components 1111. The request may be, for example, a request for a particular record within the database. After additional application-layer processing, the application components 1111 generate an external request via method invocation 1103 which is received and processed by the external system 1130. The specific format used for the external request may be based on the type of communication protocol supported by the external system 1130. Possible communication formats include (but are not limited to) remote method invocations ("RMI") or remote function calls ("RFC"). RMI is a type of remote procedure call which allows distributed objects written in Java (e.g., Java services/components) to be run remotely. RFC is a communications interface which allows external applications to communicate with R/3 systems. It should be noted, however, that the underlying principles of the invention are not limited to any particular communications protocol for communicating with an external system 1130.

Please replace paragraph [0124] with the following amended paragraph:

[0124] The external system 1130 then processes the request and provides the results back to the application components via method invocation 1104. Finally, the application components respond to the client 1100 via method invocation ~~1002~~ 1102.

Please replace paragraph [0152] with the following amended paragraph:

[0152] In an embodiment, during the registration process, a registering classfile sends method names in character string form to the dispatch unit 1702 in an order that allows the dispatch unit 1702 to properly deduce the correct numeric methodid values. For example, continuing with the above example, the "Sales" classfile 1701_1 would send method character string names in the following order: "GetMax", "UPDATE ACCOUNTING", "UPDATE BILLING". By sending the character string names in the order listed above, both the registering classfile and the dispatch unit could independently recognize that the "GetMax" method is be given a methodid value of 1; the "UPDATE ACCOUNTING" method is to be given a value of 2; and, the 'UPDATE ~~BILLIG~~ BILLING" method is to be given a value of 3.